Khái niệm OOP

OOP (Viết tắt của Objec Oriented Progamming) - lập trình hướng đối tượng là một phương pháp lập trình dựa trên khái niệm về lớp và các đôi tượng. OOP tập trung vào các đối tượng hơn là logic để thao tác chúng.

1. **Tính đóng gói (Encapsulation)**

Tính đóng gói cho phép che giấu thông tin và những tính chất xử lý bên trong của đối tượng. Các đối tượng khác đều không thể tác động trực tiếp đến dữ liệu bên trong và thay đổi trạng thái của các đối tượng mà bắt buộc phải thông qua các phương thức công khai do đối tượng đó cung cấp.

Tính chất này giúp tăng tính bảo mật cho đối tượng và tránh tình trang dữ liệu bị hư hỏng ngoài ý muốn.

2. **Tính kế thừa (Inheritance)**

Đây là tính chất được sử dụng khá nhiều. Tính kế thừa cho phép xây dụng một lớp mới (lớp Con), kế thừa và tái sử dụng các thuộc tính phương thức dựa trên lớp cũ (lớp Cha) đã có trước đó.

Các lớp Con kế thừa toàn bộ thành phần của lớp Cha và không cần phải định nghĩa lại. Lớp có có thể mở rộng các thành phần kế thừa hoặc bổ sung những thanh phần mới.

3. **Tính đa hình (Polymorphim)**

Giả sử chúng ta đã có một lớp cha và một vài lớp con kế thừa. Đôi khi chúng ta muốn sử dụng một tập hợp, ví dụ một danh sách các lớp con này. Hay chúng ta có một method riêng cho class cha, nhưng nếu ta cũng muốn sử dụng method này cho class con?

Tính đa hình đưa ra cách sử dụng một lớp con giống hệt như lớp cha để không có sự nhầm lẫn, bối rối nào giữa các dạng khác nhau. Nhưng mỗi lớp con vẫn giữ nguyên method của mình. Điều này thường xảy ra khi tái sử dụng một giao thức lớp cha. Nó đưa ra những method phổ biến, rồi mỗi lớp con thực hiện phiên bản method riêng của nó.

4. **Tính trừu tượng.**

Tính trừu tượng giống như một phiên bản mở rộng của tính đóng gói vì nó giấu đi những tính chất và phương thức cụ thể để giao thức của các đối tượng đơn giản hơn. Lập trình viên sử dụng tính trừu tượng cho vài lý do có ích khác. Nhìn chung, tính trừu tượng giúp cô lập ảnh hưởng của sự thay đổi mã code. Mục tiêu là nếu có sai sót gì xảy ra, ảnh hưởng của sự thay đổi là không nhiều.

Cú pháp Java

1. **Khai báo biến: như C++**

   int myNum;
   myNum = 15;

   Hoặc int myNum = 15;

2. **Output**

   System.out.println("Hello World!");
   System.out.print("Hello World! "); // không xuống dòng

3. **Comment: như C++**

   // Đây là comment

   /* Đây cũng là comment */

4. **If ... else: như C++**

   If (điều_kiện_1) {

   code thực thi nếu điều_kiện_1 đúng

   } else if (điều_kiện_2) {

   code thực thi nếu điều_kiện_2 đúng

   } else {

   code thực thi nếu các điều kiện kia sai

   }

   Toán tử 3 ngôi: biến = (điều kiện) ? đúng : sai

5. **Vòng lặp**

   for (khởi_tạo; điều_kiện; cập_nhật) {

   // Hành động trong vòng lặp

   }

   while (điều_kiện) {

   // Hành động trong vòng lặp

   }

   do {

```
        // Hành động trong vòng lặp
    } while (điều_kiện);
```

6. **Array:**

   String[] cars;

   Hoặc String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

7. **Method (Phương thức)**

   Một Method là một khối mã được định nghĩa trong một lớp để thực hiện một tác vụ cụ thể. Nó có thể nhận đầu vào thông qua các tham số và có thể trả về một giá trị (kiểu dữ liệu được xác định) hoặc không trả về giá trị (void). Phương thức được sử dụng để thực hiện các hành động hoặc tính toán trên đối tượng.

   <kiểu_dữ_liệu_đầu_ra> <tên_phương_thức>(<danh_sách_tham_số>) {

   // Hành động của phương thức

   return <giá_trị>; // (nếu có)

   }

   Ví dụ:

```
int addNumbers(int a, int b) {
        int sum = a + b;
        return sum;
}
```

8. **Class (Lớp)**

   Một Class là một mô hình hoặc mẫu để tạo ra các đối tượng (objects). Nó chứa định nghĩa các thuộc tính (attributes) và phương thức (methods) mà các đối tượng thuộc lớp đó có thể thực hiện.

9. **Object (Đối tượng)**

   Một Object là một thể hiện (instance) cụ thể của một lớp. Nó được tạo ra bằng cách sử dụng từ khóa "new" theo sau là tên lớp. Mỗi đối tượng có thể có các thuộc tính riêng (được gọi là instance variables) và có thể thực hiện các phương thức được định nghĩa trong lớp của nó.

10. **Instance (Thể hiện)**

    Instance là một đối tượng cụ thể của một lớp. Khi bạn tạo một đối tượng từ một lớp, bạn đang tạo một thể hiện của lớp đó.

11. **Attribute (Thuộc tính)**

Một Attribute là một biến được khai báo trong một lớp và được sử dụng để lưu trữ thông tin về trạng thái của đối tượng thuộc lớp đó. Thuộc tính cũng được gọi là các biến thành viên (member variables) hoặc trường (fields).

Ví dụ, giả sử chúng ta có một lớp "Car" (Ô tô) với các thuộc tính như "color" (màu sắc) và "speed" (tốc độ), và các phương thức như "start()" (khởi động) và "accelerate()" (tăng tốc). Khi chúng ta tạo một đối tượng từ lớp Car, chẳng hạn như "myCar", thì "myCar" sẽ là một instance (thể hiện) của lớp Car. Chúng ta có thể truy cập thuộc tính "color" của "myCar" bằng cách sử dụng "myCar.color" và gọi phương thức "start()" bằng cách sử dụng "myCar.start()".

```
class Car {
    String color;
    int speed;

    void start() {
        System.out.println("Car started");
    }

    void accelerate() {
        System.out.println("Car accelerated");
    }
}

public class Main {
    public static void main(String[] args) {
        Car myCar = new Car();
        myCar.color = "Red";
        myCar.speed = 60;

        myCar.start();
        myCar.accelerate();
    }
```

}

Trong ví dụ trên, "Car" là một lớp, "myCar" là một đối tượng (object) và "start()" và "accelerate()" là các phương thức (methods). "color" và "speed" là các thuộc tính (attributes) của đối tượng "myCar".

Tìm hiểu các lệnh git

1. **git clone**
   Git clone là một lệnh để tải xuống mã nguồn hiện có từ một kho lưu trữ từ xa (ví dụ như Github). Nói cách khác, Git clone về cơ bản tạo một bản sao giống hệt phiên bản mới nhất của một dự án trong một kho lưu trữ và lưu nó vào máy tính của bạn.
2. **git branch**
   Nhánh là một phần không thể thiếu trong Git. Bằng cách sử dụng các nhánh, các lập trình viên có thể làm việc đồng thời trên cùng một dự án. Bên cạnh đó, ta có thể sử dụng lệnh git branch để tạo, liệt kê và xóa các nhánh.
3. **git checkout**
   Đây cũng là một trong những lệnh Git được sử dụng nhiều nhất. Để làm việc trong một nhánh, trước tiên bạn cần chuyển sang nhánh đó. Mọi người chủ yếu sử dụng git checkout để chuyển từ nhánh này sang nhánh khác. Ngoài ra, chúng ta cũng có thể sử dụng nó để kiểm tra các tệp và commits.
4. **git status**

   Lệnh Git status cung cấp cho chúng ta tất cả thông tin cần thiết về nhánh hiện tại.

   Nó cho phép chúng ta xem các file được theo dõi, chưa được theo dõi và các thay đổi. Lệnh này sẽ không hiển thị bất kỳ bản ghi hoặc thông tin commit nào. Hầu hết, nó được sử dụng để hiển thị trạng thái của Git Add và Git commit.

5. **git add**

   Khi chúng ta tạo, sửa đổi hoặc xóa một tệp, những thay đổi này sẽ xảy ra trong cục bộ của chúng ta và sẽ không được đưa vào lần commit tiếp theo (trừ khi chúng ta thay đổi cấu hình).

   Chúng ta cần sử dụng lệnh git add để đưa các thay đổi của (các) tệp vào lần commit tiếp theo.

6. **git commit**

Đây là lệnh cực kỳ phổ biến và bạn không thể bỏ qua khi tìm hiểu Git là gì. Git commit có tác dụng giúp Git lưu lại một ảnh chụp màn hình/snapshot đối với các thay đổi trong thư mục trong quá trình làm việc với Git.

Với Git, khi Commit, các thay đổi sẽ được tự động lưu lại và thường nằm trong mục Staging Area. Trong hệ thống cũng sẽ lưu lại tên người chỉnh sửa để người dùng có thể dễ dàng theo dõi. Hệ thống cũng sẽ lưu trữ tên và email của người thực hiện chính sửa. Bạn cũng có thể khôi phục lại các tệp tin và chuyển sang một nhánh khác.

7. git push

Sau khi thực hiện các thay đổi của mình, điều tiếp theo bạn muốn làm là gửi các thay đổi của mình đến máy chủ từ xa. Git Push sẽ giúp tải các commit của bạn lên kho lưu trữ từ xa.

8. git pull

Lệnh git pull được sử dụng để nhận các bản cập nhật từ từ xa. Lệnh này là sự kết hợp của git fetch và git merge, có nghĩa là khi chúng ta sử dụng git pull, nó sẽ nhận các bản cập nhật từ kho lưu trữ từ xa (git fetch) và ngay lập tức áp dụng các thay đổi mới nhất trong local của bạn (git merge).

9. git revert

Đôi khi chúng ta cần hoàn tác những thay đổi mà chúng ta đã thực hiện. Chúng ta có thể thay đổi từ local hoặc từ xa (tùy thuộc vào nhu cầu của bạn), nhưng bạn phải cẩn thận sử dụng các lệnh này để tránh xóa nhầm những thứ không mong muốn.

Ưu điểm của việc sử dụng git revert là nó không động đến lịch sử commit. Điều này có nghĩa là bạn vẫn có thể xem lại tất cả các lần commit trong lịch sử của mình, ngay cả những lần xác nhận đã hoàn tác.

10. git merge

Git Merge là một lệnh dùng để hợp nhất các chi nhánh độc lập thành một nhánh duy nhất trong Git.